Des POMDPs avec des variables d'état visibles

Mauricio Araya-López, Vincent Thomas, Olivier Buffet, François Charpillet

Des POMDPs avec des variables d'état visibles

Mauricio Araya-López[1], Vincent Thomas[1], Olivier Buffet[2], and François Charpillet[2]

Nancy Université[1] – INRIA[2]
LORIA – Campus Scientifique – BP 239
54506 Vandoeuvre-lès-Nancy Cedex
prénom.nom@loria.fr — http://maia.loria.fr/

Résumé :

Les difficultés rencontrées dans les problèmes de décision séquentielle dans l'incertain sont souvent liées à la grande taille de l'espace d'états à considérer. Exploiter la structure du problème, par exemple en employant une représentation factorisée, est une approche souvent efficace mais, dans le cas des problèmes de décision markoviens partiellement observables, elle néglige un aspect important : le fait que certaines variables d'état peuvent être visibles. Dans le présent article nous proposons d'exploiter le fait que l'espace d'état peut-être factorisé en une partie visible et une partie cachée. En prenant l'exemple d'*Incremental Pruning*, nous montrons comment adapter des algorithmes classiques à cette factorisation et quels sont les bénéfices obtenus, entre autres sur la base de résultats expérimentaux.

Mots-clés : Processus de décision markoviens partiellement observables, POMDP, factorisation.

1 Introduction

La prise de décision séquentielle dans l'incertain est un important domaine de recherche. Il y a eu de nombreux travaux sur les processus de décision markoviens (MDP) (Bertsekas & Tsitsiklis, 1996) et leurs extensions comme les processus de décision markoviens partiellement observés (POMDP) (Smallwood & Sondik, 1973). Dans de nombreux cas, les difficultés sont liées à la taille de l'espace d'état (et d'action), qui souffre d'une explosion combinatoire quand la taille du problème augmente. De nombreuses approches cherchent à exploiter la structure du problème en utilisant par exemple des approximateurs de fonction ou une représentation compacte exacte lorsque c'est possible.

Nous nous concentrons ici sur les processus décisionnels de Markov, lesquels sont d'autant plus importants que de nombreux problèmes de décision sont rendus difficiles du fait de la connaissance imparfaite de l'état du système (à cause d'une observabilité partielle) comme, par exemple, le diagnostic médical, la surveillance ou la maintenance de machine (Cassandra, 1998b). Comme pour les MDP, les approches de résolution classiques sont fondées sur la programmation dynamique qui consiste à calculer la valeur attendue optimale pour chaque état (de croyance), comme pour l'algorithme *Witness* ou l'algorithme *Incremental Pruning* (Cassandra, 1998a). Un nombre important d'approches avancées se fonde sur le fait que le problème présente souvent une structure particulière et qu'il peut être modélisé de manière efficace par un POMDP factorisé (fPOMDP) (Hansen & Feng, 2000), dans lequel les états et les observations sont représentées par plusieurs variables d'états.

Dans cet article, nous remarquons que, malgré de nombreux travaux sur les POMDP factorisés, une propriété importante possédée par de nombreux problèmes a été négligée – à notre connaissance[1] –, à savoir le fait qu'une partie de l'état est directement observable. Dans ce dernier cas, le problème est situé à mi-chemin entre les MDP et les POMDP "classiques", ce qui peut être exploité pour réduire la dimensionalité de la fonction de valeur à calculer, et ainsi augmenter la vitesse de nombreux algorithmes existants.

Après avoir présenté les POMDP dans la section 2, la section 3 introduit le modèle POMDP à *variables d'état visibles* et montre dans quelle mesure il est possible d'adapter les algorithmes classiques et de les

[1]Pendant la période de préparation de la version finale de cet article, nous avons découvert l'existence de travaux récents basés précisément sur cette idée sous le nom de MDP à observabilité mixte (MOMDP) (Ong *et al.*, 2009). La principale différence est que cette idée est appliquée : dans leur cas sur des algorithmes *point-based* permettant une résolution approchée de POMDP à horizon infini, et dans le présent travail sur des algorithmes tels qu'*Incremental Pruning* permettant la résolution exacte de POMDP à horizon fini.

rendre plus efficaces. L'intérêt de cette approche est ensuite évalué empiriquement dans la section 4 avant la discussion et la conclusion.

2 Etat de l'art sur les POMDP

Un POMDP est habituellement défini par un tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, O, r, b_0 \rangle$ où, à chaque pas de temps, le système se trouvant dans l'état $s \in \mathcal{S}$ (*l'espace d'état*), l'agent effectue une action $a \in \mathcal{A}$ (*l'espace d'action*) qui a pour conséquence (1) une transition vers l'état s' selon la *fonction de transition* $T(s, a, s') = Pr(s'|s, a)$, (2) une observation $o \in \mathcal{O}$ (*l'espace d'observation*) selon la *fonction d'observation* $O(s', a, o) = Pr(o|s', a)$ et (3) une *récompense* scalaire $r(s, a)$. b_0 correspond à la distribution de probabilité initiale sur les états. Sauf indication contraire, les ensembles d'états, d'action et d'observation sont finis (Cassandra, 1998a).

Le problème consiste pour l'agent à trouver une *politique* de décision π qui choisit à chaque pas de temps la meilleure action à effectuer à partir de ses observations et de ses actions passées afin de maximiser son gain futur (qui peut être mesuré par exemple par la récompense totale accumulée ou la récompense moyenne par pas de temps). Comparé à la planification classique déterministe, l'agent doit faire face à un système non seulement doté d'une dynamique incertaine, mais aussi dont l'état courant est connu de manière imparfaite.

2.1 Belief-MDP

L'agent peut classiquement raisonner sur l'état caché du système en calculant un *état de croyance* (*belief state*) $b \in \mathcal{B} = \Pi(\mathcal{S})$ (l'ensemble de probabilité de distribution sur $\mathcal{S}$) en utilisant la formule de mise à jour suivante (basée sur la loi de Bayes) quand il effectue une action a et observe o :

$$b^{a,o}(s') = \frac{O(s', a, o)}{Pr(o|a; b)} \sum_{s \in \mathcal{S}} T(s, a, s') b(s),$$

où $Pr(o|a; b) = \sum_{s,s'' \in \mathcal{S}} O(s'', a, o) T(s, a, s'') b(s)$.

En utilisant les états de croyance, un POMDP peut être réécrit comme un MDP sur l'espace de croyance – ou *belief MDP* – $\langle \mathcal{B}, \mathcal{A}, \mathcal{T}, \rho \rangle$, où les nouvelles fonctions de transition et de récompense sont définies sur $\mathcal{B} \times \mathcal{A} \times \mathcal{B}$. Avec cette reformulation, il est possible d'étendre de nombreux résultats théoriques des MDP, tels que l'existence d'une politique déterministe optimale. Un problème réside dans le fait que même si le POMDP a un nombre fini d'états, le belief-MDP correspondant est défini sur un espace de croyance continu, et donc infini.

Pour le moment, nous ne considérons que des problèmes à horizon fini ($t \in 0..T$), un critère de performance correspondant à la récompense accumulée, et une politique prenant en entrée l'état de croyance courant. L'objectif est donc de trouver la politique optimale vérifiant $\pi^* = \arg\max_{\pi \in \mathcal{A}^{\mathcal{B}}} J^\pi(b_0)$ avec

$$J^\pi(b_0) = E\left[\sum_{t=0}^{T-1} r_t \middle| b_0, \pi \right],$$

où b_0 est l'état de croyance initial et r_t la récompense obtenue au pas de temps t. Le principe d'optimalité de Bellman (Bellman, 1954) permet de calculer cette fonction de manière récursive grâce à la *fonction de valeur*

$$V_n^*(b) = \max_{a \in \mathcal{A}} \left[\rho(b, a) + \beta \sum_{b' \in \mathcal{B}} \phi(b, a, b') V_{n-1}^*(b') \right], \quad (1)$$

où, pour chaque $b \in \mathcal{B}$, $V_0^*(b) = 0$ et $J^\pi(b) = V_{n=T}^*(b)$.

2.2 Linéarité par morceaux et convexité

Ce calcul récursif a la propriété de générer une fonction de valeur convexe, continue, et linéaire par morceaux (propriété notée "PWLC" par la suite) pour chaque horizon (Smallwood & Sondik, 1973). Ainsi, chaque fonction est définie par un ensemble d'hyperplans (chacun étant représenté par un vecteur) et la valeur pour un état de croyance donné correspond à la valeur de l'hyperplan le plus haut (cf figure 1-b). Par exemple, si Γ_n correspond à l'ensemble des vecteurs représentant la fonction de valeur pour un horizon n, alors $V_n^*(b) = \max_{\gamma \in \Gamma_n} \sum_s b(s)\gamma(s)$.

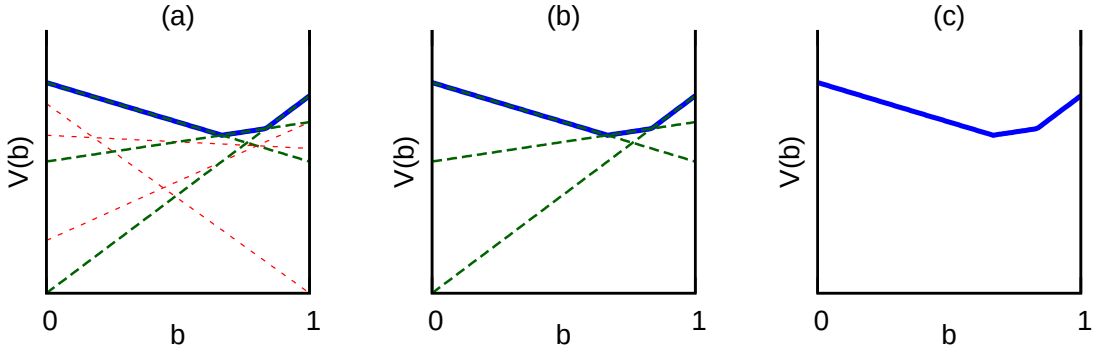

FIG. 1 – Représentation à l'aide d'un Γ-set de la fonction de valeur dans un espace de croyance à 1D avec (a) et sans (b) les hyperplans inutiles (voir sec. 2.3). (c) présente la fonction de valeur réelle.

2.3 *Incremental Pruning*

En tirant parti de la propriété PWLC, il est possible d'effectuer la mise à jour de Bellman en utilisant la factorisation suivante de l'équation 1 :

$$V_n^*(b) = \max_{a \in \mathcal{A}} \sum_o \sum_s b(s) \left[\frac{r(s,a)}{|\mathcal{O}|} + \sum_{s'} T(s,a,s') O(s',a,o) \chi_{n-1}(b^{a,o}, s') \right] \tag{2}$$

avec $\chi_n(b) = \arg\max_{\gamma \in \Gamma_n} b \cdot \gamma$.[2] Si nous considérons le terme entre parenthèses dans l'équation 2, cela génère $|\mathcal{O}| \times |\mathcal{A}|$ Γ-sets, avec une taille de $|\Gamma_{n-1}|$ pour chacun d'entre eux. Cependant, certains vecteurs γ peuvent se révéler inutiles, parce que les hyperplans correspondants se situent sous la fonction de valeur (cf fig. 1-a et b). Ces ensembles non-*parcimonieux* sont définis par :

$$\overline{\Gamma_n}^{a,o} = \{ \frac{r^a}{|\mathcal{O}|} + P^{a,o} \cdot \gamma_{n-1}, \, \forall \gamma_{n-1} \in \Gamma_{n-1} \},$$

où $P^{a,o}(s,s') = T(s,a,s')O(s',a,o)$, $r^a(s) = r(s,a)$ et $\Gamma_n^{a,o} = \text{PRUNE}(\overline{\Gamma_n}^{a,o})$.

L'algorithme d'*Batch Enumeration* de Monahan (Monahan, 1982) utilise cette mise à jour suivie par une simple phase d'élagage (retirant les vecteurs dominés) à chaque itération :

$$\Gamma_n = \text{PRUNE} \left(\bigcup_a \bigoplus_o \overline{\Gamma_n}^{a,o} \right),$$

où $\bigoplus$ est la somme croisée (*cross-sum*)[3] des Γ-sets. De la même manière, l'algorithme *Incremental Pruning* (IP) de Cassandra *et al.* (Cassandra *et al.*, 1997) — qui est plus efficace — peut être écrit :

$$\Gamma_n = \text{PRUNE} \left(\bigcup_a \text{PRUNE} \left(\bigoplus_o \text{PRUNE} \left(\overline{\Gamma_n}^{a,o} \right) \right) \right).$$

Comme pour les MDP, de nombreuses techniques de résolution ont été développées pour les POMDP, par exemple en approchant la fonction de valeur comme dans *Point-Based Value Iteration* (PBVI) (Pineau *et al.*, 2003) ou *Heuristic Search Value Iteration* (HSVI) (Smith & Simmons, 2004).

3 POMDP avec variables d'état visibles

D'un coté, le modèle classique des POMDP est essentiellement un MDP observable indirectement, parce que l'information concernant l'état est obtenue indirectement par l'intermédiaire d'observations instantanées. D'un autre coté, dans un FOMDP (*fully Observable MDP*), on accède directement à l'état courant à

[2] La fonction χ retourne un vecteur, ainsi $\chi_n(b,s) = (\chi_n(b))(s)$.

[3] La somme croisée est définie par : $\bigoplus_k R_k = R_1 \oplus R_2 \oplus \ldots \oplus R_k$, où $P \oplus Q = \{p+q | p \in P, q \in Q\}$.

chaque pas de temps. Nous proposons un scénario à mi-chemin, où on peut accéder directement à quelques variables d'états – à savoir les *variables visibles* – mais où les autres restent des *variables cachées*.

Il existe un grand nombre de problèmes pour lesquels des variables visibles et cachées coexistent dans l'état du système. En particulier, cette propriété est commune dans le domaine de l'*active sensing*. Cependant, cette structure particulière n'est pas exploitée dans le cadre des POMDP, ce qui conduit à des calculs inutiles dans plusieurs techniques de résolution.

C'est pourquoi nous proposons les vsv-POMDP, une extension des POMDP qui exploite l'existence de *variables d'état visibles* et qui peut être comprise comme un modèle hybride entre les POMDP et les MDP. Le reste de cette section présente les résultats théoriques issus de la distinction entre variables d'états visibles et cachées.

3.1 Variables visibles et cachées

Dans un POMDP standard, l'état est constitué par un ensemble de variables qui évoluent en respectant la propriété de Markov. A chaque pas de temps, une observation instantanée est produite et constitue la variable d'entrée pour l'agent. Désormais, l'état s peut se décomposer en deux variables : une variable visible s_v et une variable cachée s_h. Cette simple distinction amène des résultats intéressants pour les approches de résolution.

3.1.1 Etats et observations

Si on examine plus précisément les conséquences de cette séparation de l'état en variables visibles et cachées, on peut remarquer que les observations peuvent être elles aussi séparées en deux comme illustré par la figure 2. Dans cette factorisation, les variables visibles sont incluses de manière redondante dans l'état et dans l'observation. C'est pourquoi la contrepartie "visible" o_v de l'observation correspond à la variable s_v. D'autre part, o_w dépend de l'état global du système et de la dernière action effectuée.

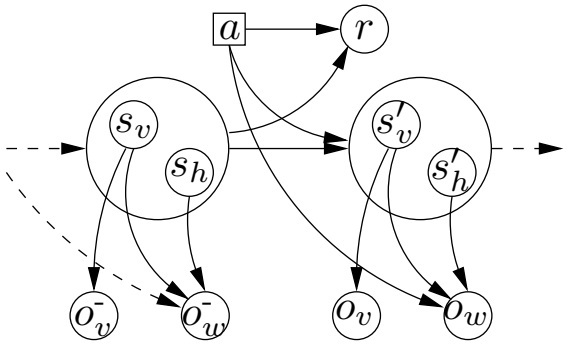

FIG. 2 – Un vsv-POMDP vu comme un réseau bayésien

De manière formelle, l'espace d'état peut être partitionné en deux : $\mathcal{S} = \mathcal{S}_v \times \mathcal{S}_h$, et il en va de même pour l'espace des observations : $\mathcal{O} = \mathcal{O}_v \times \mathcal{O}_w$. La fonction de transition reste inchangée mais peut s'exprimer à l'aide des variables visibles et cachées : $T(s, a, s') = T(s_v, s_h, a, s'_v, s'_h)$. On peut noter que la fonction de transition est désormais une distribution de probabilité jointe et peut s'exprimer à l'aide de probabilités conditionnelles et marginales sur les deux variables. De plus, si les variables visibles et cachées sont indépendantes, la fonction de transition peut s'exprimer comme la multiplication des deux modèles de transition. Les optimisations fondées sur cette propriété ont été abordées par les POMDP factorisés (cf section 5) et ne seront pas abordées dans cet article.

3.1.2 Fonction d'observation marginale

La fonction d'observation fournit une observation en fonction d'une action et d'un état du système donné. Nous savons désormais qu'une partie de l'état est dupliquée dans l'observation. Ainsi si ces parties ne correspondent pas, la probabilité de l'observation est nulle.

De manière formelle, la fonction d'observation peut s'écrire en séparant les variables d'état et d'observation $O(s', a, o) = O(s'_v, s'_h, a, o_v, o_w)$. La distribution de probabilité jointe peut aussi se séparer en deux :

$Pr(o_v, o_w|a, s'_v, s'_h) = Pr(o_v|a, s'_v, s'_h, o_w)Pr(o_w|a, s'_v, s'_h)$ ce qui peut se réécrire comme

$$O(s', a, o) = \delta(o_v, s'_v)\Omega(s'_h, s'_v, a, o_w) \tag{3}$$

où $\Omega(s'_h, s'_v, a, o_w) = \sum_{o_v} O(s'_v, s'_h, a, o_v, o_w)$ et $\delta(x, y) = \begin{cases} 1 & \text{if } x = y, \\ 0 & \text{else.} \end{cases}$

La fonction Ω correspond à la fonction d'observation marginale et fournit une observation o_w étant donnés une action a, un état caché s'_h et un état visible s'_v. Habituellement, cette fonction peut être obtenue directement à partir de la définition du problème. Si ce n'est pas le cas, elle peut être calculée à coût très faible.

3.2 Résolution des vsv-POMDP

Dans cette section, nous présentons les résultats obtenus en appliquant les idées décrites dans la section 3.1.1 aux états de croyance, à la fonction de valeur et à l'algorithme *Incremental Pruning*.

3.2.1 Etat de croyance

Comme un état de croyance est une distribution de probabilité sur les états $Pr(s; b)$, la distribution jointe $Pr(s_v, s_h; b)$ peut s'écrire comme $Pr(s_v|s_h; b)Pr(s_h; b)$. De plus, comme s_v est visible, il n'existe qu'une valeur de s_v^b pour un état de croyance b donné. Le second terme peut être compris comme l'état de croyance de la variable cachée et sera noté $b_h(s_h)$. En résumé, l'état de croyance peut s'exprimer de manière similaire à l'équation 3 sous la forme :

$$b(s) = \delta(s_v^b, s_v)b_h(s_h). \tag{4}$$

Il convient de noter que l'équation 4 représente implicitement un point de l'espace de croyance par la paire $b = (s_v^b, b_h)$ qui constitue une compression de l'état de croyance. En conséquence, la mise à jour de l'état de croyance peut être faite séparément pour s_v^b et b_h. Ainsi, b_{t+1}^{a,o_w,o_v} est calculé de la manière suivante :

$$\begin{aligned} s^b_{v,t+1} &:= o_v \\ b_{h,t+1}(s'_h) &:= Pr(s'_h|a, o_w, o_v, s'_v, s^b_{v,t}; b_{h,t}), \forall s'_h \in \mathcal{S}_h \\ &\quad (= Pr(s'_h|a, o; b)), \end{aligned} \tag{5}$$

où le terme s'_v peut être retiré de la mise à jour de l'état de croyance lié à l'état caché puisque l'équation 4 implique $s'_v = o_v$.

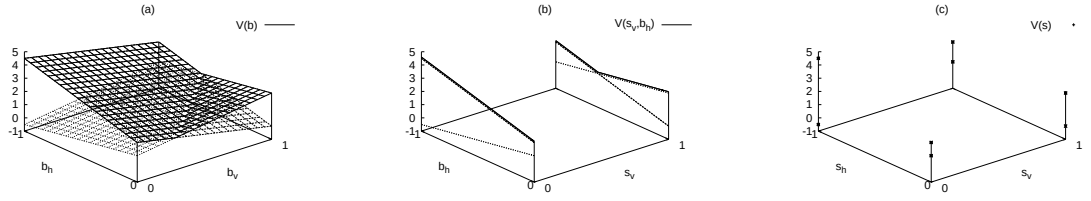

FIG. 3 – La même fonction de valeur pour un POMDP, un vsv-POMDP, et un MDP. – Note : Cette représentation est pratique mais abusive : une fonction de valeur sur 4 états nécessite un tracé en 4D.

La représentation hybride d'un point de l'état de croyance – pour moitié probabilité de distribution et pour moitié état – est la raison principale pour laquelle l'approche basée sur les vsv-POMDP est une technique à mi-chemin entre POMDP et MDP. La figure 3 propose une illustration graphique de cette idée. La première image (à gauche) montre la représentation d'un (hyper)plan d'une fonction de valeur dans un POMDP dont les variables s_v et s_h sont considérées cachées.[4] Ainsi, elles sont représentées par les variables d'états de croyance continues b_v et b_h. La troisième image (à droite) montre la représentation de la même fonction

[4]Pour des soucis de représentation, nous nous sommes limités à des états s_v et s_h correspondant à des variables binaires.

de valeur pour un MDP, où les deux variables sont directement observables. Ainsi il n'est pas nécessaire de faire appel à des états de croyance. Dans ce cas, la fonction de valeur est représentée par un ensemble discret de points (ici quatre) plutôt que par un ensemble de plans comme dans la première image. L'approche basée sur les vsv-POMDP est illustrée sur l'image centrale, dont une des dimensions correspond à l'état de croyance sur s_h comme un POMDP, mais l'autre est directement la variable s_v comme pour un MDP. En conséquence, la fonction de valeur est représentée comme un *ensemble d'ensembles d'hyperplans de plus faible dimensionalité*. Cela peut être représenté graphiquement par des tranches (ou des coupes) des hyperplans du POMDP de la figure 3.

3.2.2 Fonction de valeur

La représentation hybride de l'état de croyance et la fonction marginale d'observation présentées ci-dessus conduisent à un calcul plus rapide de la fonction de valeur en fonction des variables d'état visibles. En particulier, nous montrons comment calculer la fonction de valeur sous la forme d'un *ensemble* de représentations parcimonieuses de Γ-sets de faible dimension plutôt que sous la forme d'un ensemble unique comme dans les techniques de résolution classiques.

L'idée principale consiste à utiliser les résultats des sections précédentes pour réécrire l'équation 2 à l'aide des équations 3, 4 et 5, de la représentation hybride de $b = (s_v^b, b_h)$ et du résultat $\sum_a \delta(a,b) f(a) = f(b)$:

$$V_n^*(s_v^b, b_h) = \max_{a \in \mathcal{A}} V_n^a(s_v^b, b_h)$$

$$V_n^a(s_v^b, b_h) = \sum_{o_w} \sum_{o_v} V_n^{a,o_w,o_v}(s_v^b, b_h)$$

$$V_n^{a,o_w,o_v}(s_v^b, b_h) = \sum_{s_h} b_h(s_h) \gamma_n^{a,o_w,o_v,s_v^b,b_h} \tag{6}$$

$$\gamma_n^{a,o_w,o_v,s_v^b,b_h} = \frac{r(s_v^b, s_h, a)}{|\mathcal{O}_w||\mathcal{O}_v|} + \sum_{s'_h} \Omega(s'_h, o_v, a, o_w) T(s_v^b, s_h, a, o_v, s'_h) \chi_{n-1}(b_h^{a,o_w,o_v}, o_v, s'_h) \tag{7}$$

avec

$$\chi_n(b_h, s_v, s_h) = \arg\max_{\gamma_{s_v} \in \Gamma_n^{s_v}} \gamma_{s_v} \cdot b_h \tag{8}$$

Si, pour un POMDP normal, la fonction de valeur peut être représentée à chaque pas de temps comme un ensemble de vecteurs de dimension $|\mathcal{S}|$, dans le vsv-POMDP, la même fonction de valeur peut être représentée comme $|\mathcal{S}_v|$ ensembles de vecteurs de dimension $|\mathcal{S}_h|$. Cela s'explique en considérant l'équation 8, laquelle décrit une fonction sélectionnant un γ-vecteur à partir de l'ensemble $\Gamma_n^{s_v}$. L'indice s_v traduit le fait que, pour chaque itération de la valeur, il y a $|\mathcal{S}_v|$ ensembles de représentations parcimonieuses à considérer. Chacune de ces representations est associée à un état visible s_v donné et correspond à la fonction de valeur optimale définie sur l'espace des croyances sur s_h pour cet état visible.

Cela conduit à former un ensemble de Γ-sets, qu'on notera $\Psi_n = \{\Gamma_n^{s_v} | s_v \in \mathcal{S}\}$, où chaque $\gamma \in \Gamma_n^{s_v}$ possède $|\mathcal{S}_h|$ dimensions. En prenant en compte la structure de l'équation 6, l'ensemble Ψ_n est une manière naturelle de représenter la fonction de valeur en fonction de l'état visible et de la croyance sur la variable cachée.

Chaque $\Gamma_n^{s_v}$ de l'ensemble Ψ_n est construit indépendamment de la même manière que pour un POMDP normal. L'équation 7 génère désormais $|\mathcal{O}_w| \times |\mathcal{O}_v| \times |\mathcal{A}|$ ensembles non parcimonieux qui doivent subir une somme croisée (*cross-sum*) et être élagués pour obtenir chaque $\Gamma_n^{s_v}$. Il est important de remarquer que, pour calculer chaque $\Gamma_n^{s_v} \in \Psi_n$, on a besoin de l'ensemble Ψ_{n-1} en entier. Dans la prochaine section, nous montrerons les avantages des vsv-POMDP lors de la phase d'élagage.

3.2.3 *Incremental Pruning*

Le goulot d'étranglement computationnel des algorithmes effectuant des élagages est la résolution de programmes linéaires (LP) de grande taille. Chaque élagage d'un Γ-set non parcimonieux nécessite la résolution d'un programme linéaire (LP) avec $|\Gamma|$ contraintes. La complexité temporelle de la résolution d'un LP croît de manière spectaculaire avec la taille de Γ et la dimensionalité des vecteurs. L'équation 8 montre

clairement que les vsv-POMDP reposent sur des vecteurs de plus petite dimension, au prix d'un plus grand nombre de Γ-sets à élaguer.

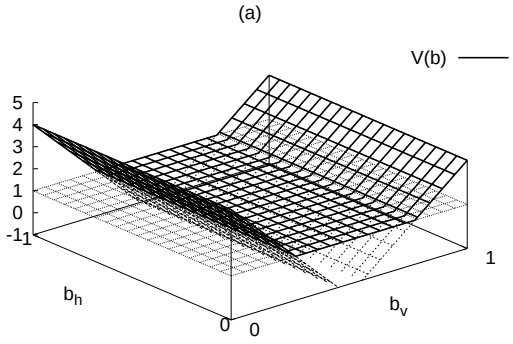

FIG. 4 – Une fonction de valeur 2D dont l'hyperplan inférieur disparaîtra dans un vsv-POMDP

De plus, les vsv-POMDP fournissent de nouvelles opportunités d'élagage de vecteurs du fait de la nature discrète des variables visibles. Dans l'image du milieu de la figure 3, le vecteur inférieur dans la tranche $s_v = 0$ est complètement dominé par les autres, et peut sans dommage être retiré de l'ensemble Γ^0. La figure 4 montre le cas particulier d'un vecteur qui fait partie d'un ensemble parcimonieux dans un POMDP, mais est complètement retiré de tous les Γ-sets dans le vsv-POMDP. Cet élagage supplémentaire de vecteurs permet de maintenir de plus petits Γ-sets, et donc d'améliorer la capacité de passage à l'échelle des algorithmes.

Nous avons ainsi suffisamment d'outils pour résoudre des vsv-POMDP en modifiant légèrement des algorithmes classiques de résolution exacte de POMDP. Nous avons choisi *Incremental Pruning* parce que c'est un algorithme de résolution exacte efficace et très connu. Nos modifications consistent en l'exécution d'un pas d'IP par valeur de l'état visible, en partant d'un Ψ-set qui représente la fonction de valeur de la dernière étape complète dans la représentation hybride de l'état de croyance. Concrètement, l'extension *vsv-IP* peut s'écrire comme suit :

$$\forall s_v \in \mathcal{S}_v, \Gamma_n^{s_v} = \text{PRUNE}\left(\bigcup_a \text{PRUNE}\left(\bigoplus_{o_v}\bigoplus_{o_w} \text{PRUNE}\left(\overline{\Gamma_n}^{a,o_v,o_w,s_v}\right)\right)\right) \tag{9}$$

La solution de vsv-IP – ou plus généralement d'un algorithme vsv avec élagage de vecteurs – est un ensemble Ψ_n de Γ-sets qui décrivent complètement la fonction de valeur. Cette représentation est équivalente à l'unique Γ-set que l'algorithme IP normal retourne. Un graphe de politique peut être déduit de l'un comme de l'autre.

4 Résultats expérimentaux

Dans cette section nous comparons la complexité empirique de la résolution d'un problème de cache-cache modélisé comme un POMDP à celle du même problème modélisé comme un vsv-POMDP. Ce problème est simplement un exemple de problème POMDP avec des variables d'état visibles.

Les expérimentations ont été conduites en employant le logiciel `pomdp-solve` de Cassandra[5] en générant automatiquement des fichiers dans le format requis (Littman *et al.*, 1995). `Pomdp-solve` supporte plusieurs algorithmes, des premiers algorithmes par énumération à des algorithmes basés sur PBVI. Ce solveur propose *Incremental Pruning* comme algorithme par défaut, que nous utilisons – sans optimisation de paramètres – pour la résolution de POMDP. Nous avons étendu ce même solveur pour implémenter vsv-IP en utilisant le même format de fichier, mais aussi en passant $|\mathcal{S}_v|$ en paramètre et en forçant un ordre pour l'encodage des variables d'état et d'observation ($\mathcal{S}_v \times \mathcal{S}_h$ pour l'état et $\mathcal{O}_w \times \mathcal{O}_v$ pour les observations).

[5] `http://www.cassandra.org/pomdp/code/index.shtml`

4.1 Le problème de cache-cache

Le problème que nous proposons implique 2 agents : un fugitif (noté h pour *hider*) et un traqueur (noté s pour *seeker*). Au départ, le fugitif et le traqueur sont placés au hasard dans un même labyrinthe. Le fugitif a une stratégie fixe (qui peut être déterministe ou stochastique) et le problème consiste à calculer le comportement optimal du traqueur.

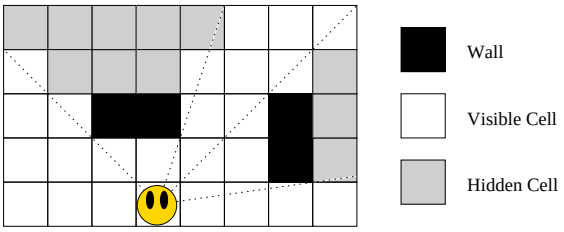

FIG. 5 – Capacité de perception de l'agent traqueur.

Le traqueur doit explorer l'environnement de manière à voir le fugitif et à maintenir un contact visuel. Les murs empêchent le traqueur de voir toutes les positions possibles du fugitif, un algorithme de lancer de rayons calculant pour chaque position du traqueur les positions qu'il peut voir (figure 5).

Le traqueur a besoin d'explorer le labyrinthe, dans l'espoir de voir le fugitif pendant cette exploration. La difficulté principale consiste à effectuer cette exploration intelligemment, en profitant des observations précédentes pour déterminer les probabilités de présence du fugitif et pour décider de la position vers laquelle aller de manière à renouveler les observations.

4.1.1 Modéliser ce problème comme un POMDP

L'environnement est un labyrinthe de $n \times m$ cellules dont b cellules vides et $w = n \cdot m - b$ murs. Il y a donc b positions possibles pour le fugitif (ou le traqueur) : $\mathcal{L} = \{l_1, l_2, \ldots, l_b\}$. Ainsi, l'espace d'état du problème peut être défini comme $\mathcal{S} = \mathcal{L} \times \mathcal{L}$, qui sont toutes les configurations possibles du fugitif et du traqueur. En d'autres termes, l'état est défini par les deux positions : $s = (l_s, l_h)$.

L'espace d'action est constitué des 8 mouvements possibles du roi d'un jeu d'échecs – un pas dans chaque direction cardinale et diagonale – plus l'action de rester dans la même cellule.

Le modèle de transition est ainsi défini par un ensemble de 9 matrices où $\mathbf{T}^a : \mathcal{S} \times \mathcal{S} \rightarrow [0, 1]$ représente la transition jointe du traqueur et du fugitif pour une action a donnée.

L'espace d'observation est défini de manière similaire à l'espace d'état, et correspond aux positions du chercheur et du fugitif. Mais, comme la position du fugitif n'est pas toujours connue, la valeur $l_{inconnu}$ a été ajoutée à l'espace $\mathcal{L}^*$ des positions du fugitif. L'espace d'observation peut alors être formellement défini par $\mathcal{O} = \mathcal{L} \times \mathcal{L}^*$ où $o = (l_s, l_h^*)$.

Le modèle d'observation peut être construit en utilisant seulement la topologie de l'environnement, la matrice $\mathbf{O} : \mathcal{S} \times \mathcal{O} \rightarrow \{0, 1\}$ représentant les observations possibles en fonction de l'état courant. Ce modèle est déterministe, de sorte que chaque état est associé à une unique observation possible. Si le traqueur peut voir le fugitif, l'observation est identique à l'état et est constituée des positions du traqueur et du fugitif. Sinon, alors la position du traqueur est connue, mais la position observée du fugitif est $l_{inconnu}$.

4.1.2 Modéliser ce problème comme un vsv-POMDP

La conversion du problème cache-cache d'un POMDP à un vsv-POMDP est simple. En effet, la modélisation vsv-POMDP de ce problème est plus naturelle que sa version POMDP. La raison en est que l'espace d'état et l'espace d'observation sont naturellement partitionnés, la position du fugitif étant évidemment cachée, et celle du traqueur étant visible. Ainsi, la variable d'état est $s = (s_v, s_h) = (l_s, l_h)$, la variable d'observation $o = (o_v, o_w) = (l_s, l_h^*)$, et la fonction Ω est une fonction (de la topologie de l'environnement) légèrement différente, la variable o_v n'étant pas prise en compte parce que donnée par s'_v. L'état de croyance est alors constitué de la position courante du traqueur, plus la croyance sur la position du fugitif.

4.2 Descriptions des expérimentations

Les cartes utilisées dans les expérimentations qui suivent sont basées sur des caractères (lettres et chiffres), les cellules vides d'une carte formant la structure d'un caractère (voir figure 6). Ces cartes sont très simples en tant que labyrinthes, mais calculer une politique optimale associée est non trivial, entre autres parce qu'elle dépend de la stratégie du fugitif.

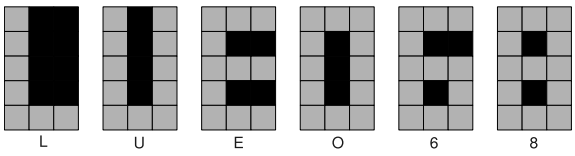

FIG. 6 – Les topologies L, U, E, O, 6 et 8 qui ont été utilisées (ici sur une grille 5×3).

Pour ces expériences nous avons employé trois stratégies différentes pour le fugitif. La première est une stratégie *statique* dans laquelle le fugitif ne bouge jamais. La seconde est une stratégie *aléatoire* dans laquelle le fugitif effectue une marche aléatoire à travers le labyrinthe sans tenir compte de la position du traqueur. Et la troisième est une stratégie *réactive* omnisciente dans laquelle le fugitif fuit de manière déterministe la position – toujours connue – du traqueur. Les positions initiales du traqueur et du fugitif sont échantillonnées uniformément sur la carte.

Nous avons sélectionné l'ensemble d'expérimentations suivantes à effectuer :

Influence de l'horizon – Pour étudier l'impact de l'horizon du POMDP sur le temps de résolution et la taille de la solution, trois topologies ont été choisies : L, U et O. Deux versions différentes de chaque topologie ont été testées : une version 3×3 et une version 4×4. Ces six cartes différentes ont été testées avec des horizons de longueurs 10, 50 et 100.

Influence de la taille de la carte – Pour cette expérience, une topologie simple et une topologie complexe ont été choisies, respectivement L et 8. Pour ces deux topologies, quatre tailles de cartes différentes ont été testées : 3×5, 5×7, 7×9 et 9×11. L'objectif est d'explorer l'impact de l'accroissement de la taille des topologies.

Influence de la topologie – Nous nous sommes concentrés sur 6 topologies différentes nommées, par ordre croissant de complexité, L, E, U, O, 6 et 8 (voir figure 6).

Les topologies L, E et U correspondent à des problèmes simples puisque la politique optimale consiste seulement à croiser le couloir ou rejoindre un coin, que le fugitif bouge ou non. Le problème devient plus complexe avec les topologies O, 6 et 8 puisqu'elles contiennent des boucles qui permettent au fugitif d'échapper à la vue du traqueur, en particulier quand la taille de la carte est grande.

4.3 Résultats

Pour chaque expérience plusieurs statistiques ont été collectées – telles que le temps, le nombre de LP, le nombre de contraintes, le nombre de vecteurs, etc. – toutefois nous ne présenterons que les deux plus significatives : le *temps* et la *taille de la solution*.

Le *temps* est le temps de calcul empirique pris par le solveur de Cassandra pour calculer la fonction de valeur optimale pour une expérience donnée. La limite de temps pour ces expériences était de 7200 secondes (2 heures), ce qui veut dire que, si la solution n'était pas trouvée après 2 heures, le solveur était arrêté intentionnellement.

La *taille de la solution* est le nombre d'éléments qu'un fichier solution contient. La taille de la solution est $|\Gamma_n| \cdot (|\mathcal{S}| + 1)$ dans le cas de la modélisation POMDP, alors que, dans le cas vsv-POMDP, elle correspond à $\sum_{s_v} |\Gamma_n^{s_v}| \cdot (|\mathcal{S}_h| + 1)$. Le $+1$ représente le fait que, pour chaque vecteur, la solution fournit aussi une valeur désignant une action à effectuer.

Pour toutes les expérimentations, nous avons décidé de ne montrer que le fugitif avec une stratégie aléatoire, parce que la nature des résultats est similaire pour toutes les stratégies.[6]

La table 1 présente les résultats pour les expérimentations sur l'horizon, fournissant un très bon résumé de cette section toute entière puisqu'elle inclut aussi des variations de la topologie et de la taille de la carte.

[6]Les résultats sont légèrement différents en termes d'échelle et de forme, mais les conclusions restent les mêmes.

Dans cette table, il est évident que la taille de la solution ne change pas avec l'horizon ni avec IP, ni avec vsv-IP. La raison est que la structure optimale de vecteurs γ devient stable très rapidement dans des problèmes pour de petits espaces d'état (3 ou 4 étapes pour la plupart des topologies). Il est naturel de s'attendre à ce que la taille des solutions croisse avec la complexité et la taille de la carte, toutefois la croissance de IP est considérablement plus forte que celle de vsv-IP.

TAB. 1 – Temps d'exécution et taille de la solution en fonction de l'horizon, de la topologie et de la taille de la carte, pour un fugitif utilisant la stratégie *aléatoire*

Carte	**Horizon**	**IP**		**vsv-IP**		**Accélération**
		Temps [sec]	Taille Sol.	Temps [sec]	Taille Sol.	
L-**3**×**3**	10	0,15	52	0,06	30	2,50
	50	0,84	52	0,34	30	2,47
	100	1,67	52	0,66	30	2,53
L-**4**×**4**	10	0,56	100	0,22	56	2,56
	50	2,95	100	1,17	56	2,52
	100	5,96	100	2,15	56	2,77
U-**3**×**3**	10	2,72	600	1,06	144	2,57
	50	15,31	600	5,93	144	2,58
	100	31,21	600	11,95	144	2,61
U-**4**×**4**	10	40,88	5151	4,05	374	10,09
	50	272,63	5151	24,83	374	10,98
	100	561,04	5151	50,86	374	11,03
O-**3**×**3**	10	587,16	49920	5,62	576	104,48
	650	3407,72	49920	32,34	576	105,37
	100	6954,45	49855	65,48	576	106,20
O-**4**×**4**	10	>7200,00	–	35,50	2392	>202,82
	50	>7200,00	–	213,87	2392	–
	100	>7200,00	–	438,24	2392	–

Le temps d'exécution est aussi en général plus petit pour vsv-IP, la colonne *accélération* montrant combien de fois plus vite va l'algorithme vsv-IP. Pour des problèmes simples vsv-IP va deux fois plus vite que IP et, quand les problèmes deviennent plus complexes (en taille et topologie), vsv-IP se comporte encore mieux, avec des améliorations de plusieurs ordres de magnitude. Dans le cas de O-4×4, l'IP normal n'a pas terminé avant les deux heures allouées, même pour un horizon de 10, alors que vsv-IP retourne une solution en quelques minutes pour un horizon de 1000.

L'expérience suivante explore l'influence de la taille de la carte pour la topologie L. Les résultats correspondants sont montrés dans la table 2 et la figure 7. Même si le temps n'est pas une fonction linéaire de la complexité (la figure 7 a une échelle logarithmique), l'accélération s'accroît pour de grandes cartes, ce qui veut dire que vsv-IP passe mieux à l'échelle avec la taille de la carte. La taille de la solution est meilleure pour vsv-IP, mais représentant seulement la moitié de la taille de la solution d'IP. Ce phénomène est plutôt constant, du moins pour cette expérience. Cette expérience a aussi été conduite pour la topologie 8, mais IP n'a pas terminé même pour une taille de 3×5, alors que vsv-IP a terminé pour tous les essais.

TAB. 2 – Temps d'exécution et taille de la solution en fonction de la taille de la carte L, pour un fugitif utilisant la stratégie *aléatoire* et un horizon de 10

Carte	**IP**		**vsv-IP**		**Accélération**
	Temps [sec]	Taille Sol.	Temps [sec]	Taille Sol.	
L-**3**×**5**	0,58	100	0,22	56	2,64
L-**5**×**7**	4,64	244	1,51	132	3,07
L-**7**×**9**	23,06	452	6,11	240	3,77
L-**9**×**11**	90,76	724	20,23	380	4,49

Les résultats du dernier jeu d'expériences, avec des complexités de topologies croissantes, sont présentés dans la table 3. Comme pour la topologie O-4×4 de la première expérience ou la topologie 8 de la seconde, ici l'algorithme IP normal ne s'est pas terminé pour les topologies E, U, O, 6 et 8, alors que vsv-IP terminait dans un temps raisonnable. La figure 8 montre – aussi en échelle logarithmique – que le temps de calcul

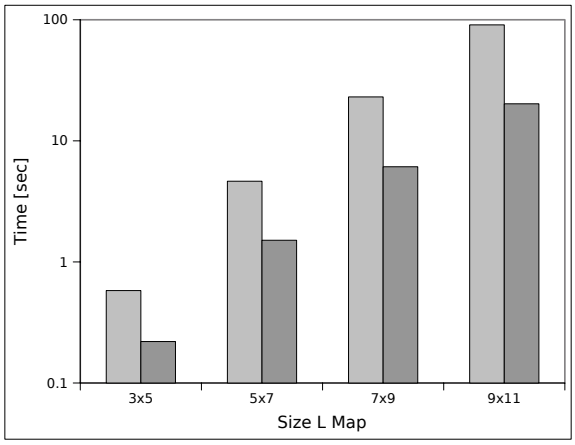

FIG. 7 – Le temps en fonction de la taille de la carte L, le fugitif ayant une stratégie *aléatoire*

croît beaucoup moins pour vsv-IP quand les cartes sont de plus en plus complexes, ce qui veut dire que notre approche passe aussi mieux à l'échelle avec la complexité inhérente au problème. La croissance de la taille de la solution est aussi bien moindre pour vsv-IP, ce qui est confirmé par la table 1.

TAB. 3 – Le temps d'exécution et la taille de la solution en fonction de la complexité de la topologie, pour un fugitif employant une stratégie *aléatoire* et un horizon de 10

Carte	**IP**		**vsv-IP**		**Accélération**
	Temps [sec]	Taille Sol.	Temps [sec]	Taille Sol.	
L-3×5	1,02	130	0,37	72	2,76
U-3×5	94,95	7395	6,64	468	14,30
E-3×5	>7200	–	19,26	1110	>373,83
O-3×5	>7200	–	80,00	4200	–
6-3×5	>7200	–	197,87	15360	–
8-3×5	>7200	–	687,51	47328	–

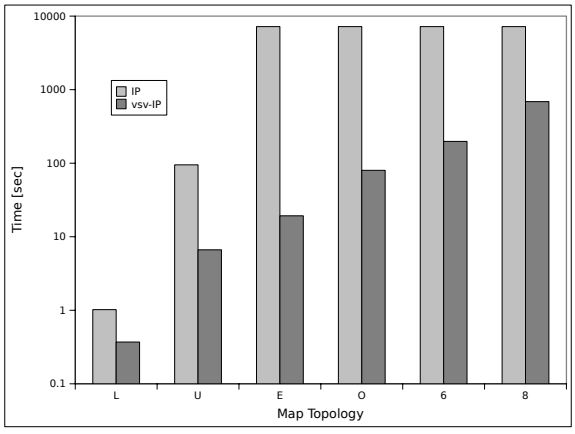

FIG. 8 – Le temps d'exécution en fonction de la complexité de la topologie, en utilisant un fugitif avec une stratégie aléatoire

En résumé, pour tous les horizons, topologies et tailles de cartes, vsv-IP a des performances significativement meilleures que l'algorithme IP original en termes de temps de calcul et de consommation de mémoire. Notre approche passe aussi mieux à l'échelle avec une complexité croissante en termes de taille et de topologie de la carte.

5 Discussion et travaux futurs

Les modifications faites sur IP pourraient aussi être appliquées à des algorithmes tels que *Witness* ou *Batch Enumeration*, vraisemblablement avec la même efficacité. Une question importante est de déterminer si cela s'étend à des algorithmes plus avancés. Considérons par exemple trois types d'algorithmes :

POMDP Factorisés – De premiers algorithmes candidats sont ceux qui reposent sur des POMDP factorisés, tels que Symbolic DP (Hansen & Feng, 2000), Symbolic Perseus (Poupart, 2005) ou Symbolic HSVI (Sim *et al.*, 2008). Cette factorisation est différente de la nôtre : elle représente aussi bien les états que les observations avec de multiples variables, les fonctions de transitions, d'observation et de récompense étant modélisées comme des réseaux bayésiens (dynamiques). En utilisant par exemple des diagrammes de décision algébriques (ADD), cela permet des calculs très efficaces. Cette factorisation plus commune est donc complémentaire de notre approche, de sorte que les deux pourraient être employées ensemble avec des bénéfices cumulés.

Techniques à base de points – Les algorithmes POMDP à base de points, tels que PBVI (Pineau *et al.*, 2003) ou Perseus (Spaan & Vlassis, 2005), utilisent un ensemble d'états de croyance représentatifs pour construire une approximation de la fonction de valeur. Ces états de croyance sont tous atteignables puisqu'ils sont échantillonnés en employant la dynamique de transition. Ainsi, les algorithmes PB (*Point-Based*) focalisent déjà sur les coupes utiles. Toutefois, ils utilisent des vecteurs Γ définis sur toutes les dimensions de l'espace de croyance. Notre technique de réduction de la dimensionalité peut donc être appliquée, avec un gain attendu en vitesse de calcul dans les mises à jour de Bellman et dans l'élagage (notons que PBVI ne requiert pas de résoudre des programmes linéaires).

Techniques en ligne – Les techniques en ligne (Ross *et al.*, 2008) reposent sur des simulations de Monte-Carlo pour estimer la fonction d'action-valeur à l'état de croyance courant. Comme pour les approches PB, elles échantillonnent des états de croyance, lesquels sont nécessairement atteignables. Une différence est que la fonction de valeur est représentée en employant les points échantillonnés, et non avec des approximateurs multi-dimensionnels. En conséquence notre approche n'est pas applicable dans ce cas.

D'autres techniques exploitent la structure du problème à résoudre, telles que des symétries (Kim, 2008), des POMDP permutables (Doshi & Roy, 2008) ou épistémiques (Sabbadin *et al.*, 2007). Certaines de ces approches – y compris la nôtre – pourraient être combinées pour bénéficier de leurs apports respectifs. Nos travaux futurs devraient se concentrer non seulement sur de telles combinaisons, mais aussi sur la recherche d'autres structures pertinentes à exploiter dans les POMDP, idéalement pour généraliser les travaux passés dans ce domaine.

Une propriété intéressante du calcul de la fonction de valeur dans les vsv-POMDP est que chaque tranche (ou coupe) peut être calculée indépendamment en utilisant la représentation précédente par Ψ-sets de la fonction de valeur. Ainsi, les ensembles Γ^{s_v} peuvent être calculés simultanément sans difficulté. Cela présente une importante opportunité de parallélisation, où chaque nœud peut calculer une coupe de la prochaine fonction de valeur indépendamment.

6 Conclusion

Dans cet article nous discutons un fait important – bien que négligé – sur les POMDP : dans de nombreux problèmes, une partie de l'état est complètement observable, de sorte qu'une large part de l'espace des états de croyance est inutile. Nous avons montré comment des algorithmes tels qu'*Incremental Pruning* peuvent être modifiés pour prendre en compte ces variables d'état visibles. Nos expérimentations confirment le gain attendu en complexité temporelle. Cette propriété est d'autant plus intéressante qu'elle peut être exploitée dans de multiples algorithmes de résolution de POMDP, mais de différentes manières toutefois. Plusieurs directions de recherche restent donc ouvertes.

Références

BELLMAN R. (1954). The theory of dynamic programming. *Bull. Amer. Math. Soc.*, **60**, 503–516.

BERTSEKAS D. & TSITSIKLIS J. (1996). *Neurodynamic Programming*. Athena Scientific.

CASSANDRA A. (1998a). *Exact and approximate algorithms for partially observable Markov decision processes*. PhD thesis, Providence, RI, USA.

CASSANDRA A. (1998b). A survey of POMDP applications. In *AAAI Fall Symposium*.

CASSANDRA A., LITTMAN M. & ZHANG N. (1997). Incremental pruning : A simple, fast, exact method for partially observable Markov decision processes. In *Proc. of the 13th Conference on Uncertainty in Artificial Intelligence (UAI)*, p. 54–61.

DOSHI F. & ROY N. (2008). The permutable POMDP : fast solutions to POMDPs for preference elicitation. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, p. 493–500, Richland, SC.

HANSEN E. & FENG Z. (2000). Dynamic programming for POMDPs using a factored state representation. In *Proceedings of the International Conference on Artificial Intelligence Planning and Scheduling (AIPS)*.

KIM K.-E. (2008). Exploiting symmetries in pomdps for point-based algorithms. In *AAAI'08 : Proceedings of the 23rd national conference on Artificial intelligence*, p. 1043–1048 : AAAI Press.

LITTMAN M. L., CASSANDRA A. R. & KAELBLING L. P. (1995). *Learning policies for partially observable environments : Scaling up*. Rapport interne.

MONAHAN G. (1982). A survey of partially observable Markov decision processes. *Management Science*, **28**, 1–16.

ONG S., PNG S., HSU D. & LEE W. (2009). POMDPs for robotic tasks with mixed observability. In *Proceedings of Robotics : Science and Systems V (RSS'09)*.

PINEAU J., GORDON G. & THRUN S. (2003). Point-based value iteration : An anytime algorithm for POMDPs. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI)*.

POUPART P. (2005). *Exploiting Structure to Efficiently Solve Large Scale Partially Observable Markov Decision Processes*. PhD thesis, University of Toronto.

ROSS S., PINEAU J., PAQUET S. & CHAIB-DRAA B. (2008). Online planning algorithms for POMDPs. *Journal of Artificial Intelligence Research (JAIR)*, **32**, 663–704.

SABBADIN R., LANG J. & RAVOANJANAHARY N. (2007). Purely epistemic markov decision processes. In *AAAI'07 : Proceedings of the 22nd national conference on Artificial intelligence*, p. 1057–1062 : AAAI Press.

SIM H., KIM K.-E., KIM J., CHANG D.-S. & KOO M.-W. (2008). Symbolic heuristic search value iteration for factored POMDPs. In *Proceedings of the Twenty-Third National Conference on Artificial Intelligence (AAAI)*.

SMALLWOOD R. & SONDIK E. (1973). The optimal control of partially observable Markov decision processes over a finite horizon. *Operation Research*, **21**, 1071–1088.

SMITH T. & SIMMONS R. (2004). Heuristic search value iteration for POMDPs. In *Proceedings of the International Conference on Uncertainty in Artificial Intelligence (UAI)*.

SPAAN M. & VLASSIS N. (2005). Perseus : Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, **24**, 195–220.